Goals and Business Rules as the Central Design Mechanism

Gil Regev [1], Alain Wegmann [1]

1 Institute for computer Communication and Application (ICA)
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
http://icawww.epfl.ch
{gil.regev, alain.wegmann}@epfl.ch

Abstract: Current UML based behavioral requirements focus on the specification of use cases. Business rules are seen as a very important part of the requirements but are either considered as a separate entity or, at best, attached to use cases. Some authors have recognized the need to include business rules within use cases but there are no concrete guidelines on how to do it. In this paper we propose a technique that uses business rules as the driving mechanism behind the design process. By explicitly defining business rules in collaborations of a group of actors, we are able to find the policies governing the behavior of the different actors involved. When we decide to consider one of these actors as the system to be designed, these policies enable us to define the goals of this system. These goals are implemented as use cases which themselves are governed by business rules for the system. When considering this system as a set of parts we loop back to the beginning of our process but at a different organizational level.

Introduction

"The aim of an enterprise description is to identify the actions that are required by the policies governing the communities that are specified by it. Such actions are concerned with the placing and fulfilling of obligations (e.g. requesting delivery; making a delivery), and the permitting or forbidding of actions (e.g. authorising or rejecting access to system facilities)." [6, part 1 p. 19]

Customers, organizations, employees, and most other entities all have goals that they strive to satisfy. In this drive to satisfy explicit and implicit goals, these actors are constrained by rules that are usually called business rules in requirements engineering.

Goals and business rules are often implicit and ingrained within the organizational behavior. In other words we don't know why we pursue some goals and why we pursue them the way we do. The reasons for their existence have vanished. Much of a development team's work is spent on making these goals and rules explicit as explained in RM-ODP.

While goals are seen as part of the functional requirements, business rules are generally viewed as being part of the non-functional requirements [14] [2] [7]. Business rules do, however, greatly influence the functional requirements (e.g. the way the system behaves). As noted by [14, p. 95] "*The business rules are management prescriptions that transcend and guide the day-to-day business decisions. Naturally any product that you build must conform to the business rules*." If business rules are used to guide decisions within an organization, why couldn't they also be used to guide a development team on the way to design the product, thereby insuring that the product conforms to these rules? Business rules directly influence the way the product behaves and functions but are normally considered to be outside the functional requirements. If business rules are defined outside of the functional requirements and are not mapped to them, how can we insure that the product conforms to these rules? Kilov [8] states that business rules are often not explicitly stated and are implicitly formalized in the application code. We can say the same for functional requirements based on use cases. Business rules are often diluted in the narrative or the exceptions section of a use case. When business rules are documented, it is usually as a list of rules within use cases or an external document. While this is better than no list at all, it does not help in understanding what rule influences what behavior.

So there are two main reasons to design systems based on goals and rules:

1. Goals and rules should be central for guiding the decisions taken during design, thus ensuring that the system fulfils its purpose.
2. Rules are subject to change as the organization in which they are specified changes due to internal changes and changes in its environment.

Without a specific binding between business rules and goals we are not able to make changes to the behavioral requirements and hence to the systems we build when these rules change.

In this paper we propose a modeling approach that integrates business rules with behavioral requirements. This approach considers the goals of the actors and the business rules that govern their behavior as the central mechanism behind the design of the system. The approach is based on theories and principles from the fields of Systems Theory, Activity Theory, Requirements Engineering and Software Engineering.

Problem Definition

Behavioral requirements in UML heavily rely on use cases. A use case is defined in UML as follows: [16, p. 2-122]: *The use case construct is used to define the behavior of a system or other semantic entity without revealing the entity's internal structure. Each use case specifies a sequence of actions, including variants, that the entity can perform interacting with actors of the entity.*

This definition doesn't take into account the fact that the systems we build need to help their stakeholders to achieve a certain number of goals. Without explicitly stating what the goals of the stakeholders are, it is easy to focus on the details of the interaction between the actors. Focusing on these interactions makes it difficult to formulate alternative design choices and weigh their pros and cons. We usually dive into the first idea we have refining it until we have a running product and not challenging the reasons for selecting this idea as the one to implement. What's more, without an explicit goal statement, it is very difficult to make the rules that constrain the achievement of the goal explicit. We then lose another dimension in our design, which is why we do things the way we do them. The all too frequent answer is "because we've always done them this way" which is not a suitable answer when we build a new product.

Several approaches have been defined in recent years to overcome the shortcomings of use cases seen only as a sequence of interactions. Essential use cases [3] capture the essence of the interactions between the user and the system. By defining these essential actions rather than the individual steps as often done in traditional use cases, essential use cases help developers to focus on users' goals and formulate better ways to achieve them. Cockburn [2] adds to essential use cases the dimension of protecting stakeholders' interests. Supporting actors in fulfilling their goals while protecting stakeholders' interests is the main reason for an enterprise information system's existence. These approaches focus on goals and don't integrate business rules into use cases making it difficult to understand why we attribute certain goals to the different actors. Gottesdiener [5] argues that "*use case narratives – the flow of events, as the UML people would say – are all too often missing the very essence of the use case, because behind every use cases are business rules at work*." Gottesdiener defines the problem but doesn't formulate a detailed solution for embedding business rules in use cases. So the important link between business rules and goals has not been achieved.

Another problem with use cases is their modeling of the system as an implicit entity making them awkward to use in situations where multiple systems interact [18]. The Catalysis method [4] defines joint actions, which bring together multiple actors called participants who can be seen as collaborating towards the achievement of a common goal. Joint actions are defined in Catalysis as "*a change in the state of some number of participant objects without stating how it happens and without yet attributing the responsibility for it to any one of the participants*" [4, p. 158]. Joint actions are a good means for defining the essence of the collaboration between multiple participating actors. However, Joint actions are not a good construct for

defining behavior based on participants' goals. The goal of each participant in a joint action is potentially different from the goals of the other participants. The joint action, however, has only one name and can thus convey at most one of the participants' goal or else be named as a noun not representing any goal. A simple example is that of a sale. The vendor's goal is to sell but the customer's goal is to buy. Joint action does not exist within the context either the buyer or the vendor, which forces the modeler to describe it as a Sell, or a Buy, or a Sale. By doing this, we either define the collaboration from the point of view of one of the participants or we loose the view of the goals of the different participants.

Two other approaches are worth mentioning with regard to goals and business rules. Goal-Oriented modeling [12] [1], attempts to bridge between non-functional and functional requirements by defining goals (or soft goals in this instance) and sub-goals that drive the design process. Little is said, though, on the fact that goals and sub-goals are actually the result of specific business rules. A complementary approach is Policy Driven Design [15] which views the functional requirements as being driven by policies alone.

Our design approach takes a dual stance in proposing to consider the interaction between goals and policies as the main design driver.

Theoretical Foundations

Finality

Teleology (from Greek telos, "end"; logos, "reason"), explanation by reference to some purpose or end; also described as final causality, in contrast with explanation by efficient causes only. Human conduct, insofar as it is rational, is generally explained with reference to ends pursued or alleged to be pursued; and human thought tends to explain the behaviour of other things in nature on this analogy, either as of themselves pursuing ends, or as designed to fulfill a purpose devised by a mind transcending nature. The most celebrated account of teleology was that given by Aristotle when he declared that a full explanation of anything must consider not only the material, the formal, and the efficient causes, but also the final cause--the purpose for which the thing exists or was produced.
Encyclopedia Britannica

Von Bertalanffy [17, p. 77-80] discusses different types of finality or purposiveness concluding that *"true finality or purposiveness meaning that the actual behavior is determined by the foresight of the goal […] is characteristic of human behavior."* Human made organizations embody such purpose and are goal seeking entities. Organizations are systems, themselves made of sub-systems, each having its own potential finality. There may be n levels of goal oriented sub-systems. A goal is generally but a sub-goal of an *overall* goal or it can be inscribed in the overall finality of the actor. As we will see later this characteristic of a system helps us identify the steps needed to achieve a system's overall goal.

When we design systems it is critically important to explicitly state the purpose not only of the system and its actors but also of the organization within which the system will operate. Since we define that systems are made out of subsystems, it follows that any system that we might consider is also a part of a bigger system and this bigger system has it's own purpose.

The implications for IT system design is that when we specify the requirements of an IT system, we need not only consider the purpose of the system we are designing but also the purpose of the organization within which it will operate and the purposes of all actors interacting with the system.

Constructivism and its Implications

Most methodologies talk about the need to *discover* requirements, intentions, goals, business rules etc. It is our view that discovering (and its synonyms) does not adequately represent the requirements elicitation

process. Discovering requirements is an objectivist worldview that implies that requirements are present in the world and need to be uncovered. To some extent this is true but assuming a constructivist worldview, we say that requirements are not only discovered but also created during the process of elicitation. Furthermore, the different stakeholders will build their understanding of what needs to be built during the process of building the system and thus create a new system with requirements that may not have existed before. The implications are that in the act modeling a system we also modify the system. Hence the importance of guiding our decisions through the goals of the stakeholders and the business rules.

Constructivism also holds that different people develop different models of the world. Extending this principle to the domain of systems, we can say that each system has its own view of itself and its environments. This in turn leads to different goals pursued by different systems. The implications to our design approach is that we need to define the view of each actor, or sub-system in our system of interest so as to understand its goals and the constraints under which it operates.

Activity Theory

Activity Theory, abbreviated AT, [9] gives a framework for modeling human behavior, both individual and collective, within the context of an activity. An activity is seen as occurring between 3 main entities: Subject, object, and Community. The subject (individual or collective) transforms the object through conscious and purposeful actions towards a desired outcome with the help of tools. Tools are said to mediate the relationship between the subject and the object. The relationship between community and subject is mediated by rules, and that between community and object is mediated by the division of labor. Thus, an activity has a purpose, which is the outcome of the transformation of the object of the activity. This is the "external" view of an activity. See Figure 1.

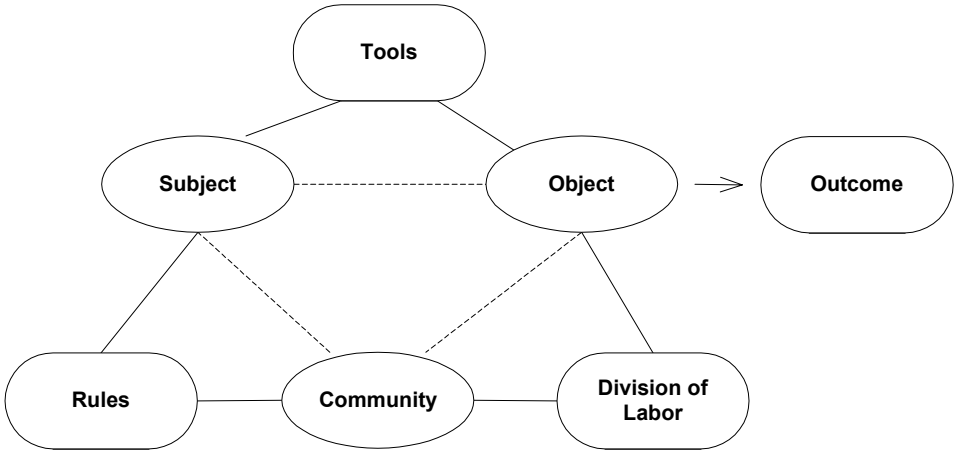

Fig. 1. External view of an activity. Adapted from [9]. This is not a UML model!

The rules that mediate the relationship between community and subject are not only rules imposed by the community. Actors also have internal rules that constrain their behavior with their tools and with the community. If we view an IT system as a tool, we see that it has to support the transformation of the object of the activity towards an outcome defined by the subject and the community whose relationship is mediated by rules. Thus the IT system has to embody the rules while supporting the "production" of the outcome.

The following table freely translates the concepts of activity theory into the UML vocabulary.

AT	UML
Subject	Actors
Community	Stakeholders
Tools	System
Object	The work to be done
Rules	Business rules
Division of Labor	Decomposition
Outcome	Outcome

Table 1. Activity Theory concepts translated to UML

The "internal" view of an activity describes its relation to actions and operations by viewing the degree to which behavior is internalized. Thus an activity is seen as having a motive, which may be long term while actions have immediate goals but are still the result of conscious planning. Operations on the other hand are seen as internalized actions triggered by specific conditions (Figure 2). AT recognizes that the boundaries between activity, action, and operation are very fuzzy. When we face problems with our environment, an operation may get "promoted" to the level of an action or even an activity while we are continually learning to transform activities into actions and into operations.

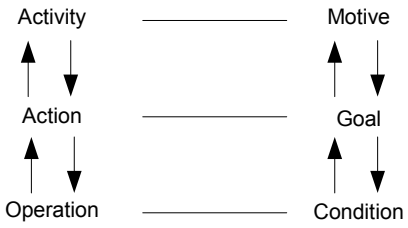

Fig. 2. Internal view of an activity as described by activity theory, Adapted from [9]

Activity theory gives a social and psychological view of an activity. In the RM-ODP section we will see a more technical discussion of activities. Activity theory helps us introduce the rules and goals into the discussion of an activity which is missing from the RM-ODP view.

RM-ODP

Business rules are the organization's response to the constraints imposed on it by its environment (e.g. by its external stakeholders) and by its own internal culture and structure. As we've seen in our discussion of Activity Theory, an actor's activity is constrained by the business rules imposed on the actor by the actor's environment (e.g. it's stakeholders, interacting systems etc) and the actor's internal rules.

A policy is defined in RM-ODP [6, p. 11] as: "*A set of rules related to a particular purpose. A rule can be expressed as an obligation, a permission or a prohibition. NOTE - Not every policy is a constraint. Some policies represent an empowerment.*"

And in the Merriam-Webster dictionary:

1 a : prudence or wisdom in the management of affairs b : management or procedure based primarily on material interest

2 a : a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions b : a high-level overall plan embracing the general goals and acceptable procedures especially of a governmental body

This makes policy a very suitable synonym for the term business rule. It also shows that policies (or business rules) and actions are intimately related hence our choice of policy as our modeling artifact.

RM-ODP [6] defines the concept of activity as "*A single-headed directed acyclic graph of actions.*" An activity can be seen as an atomic concept having one possible outcome and which is devoid of internal steps, in which case we simply refer to it as an activity. The same activity can also be seen as a set of interacting activities, we then call it a composite activity and its internal interacting activities are called component activities [19]. This specification of an activity is analogous to the specification of an object as an atomic object or a composite object made out of a set of interacting component objects. Figure 3 shows these two dual views of object vs. composite object and Activity vs. composite activity.

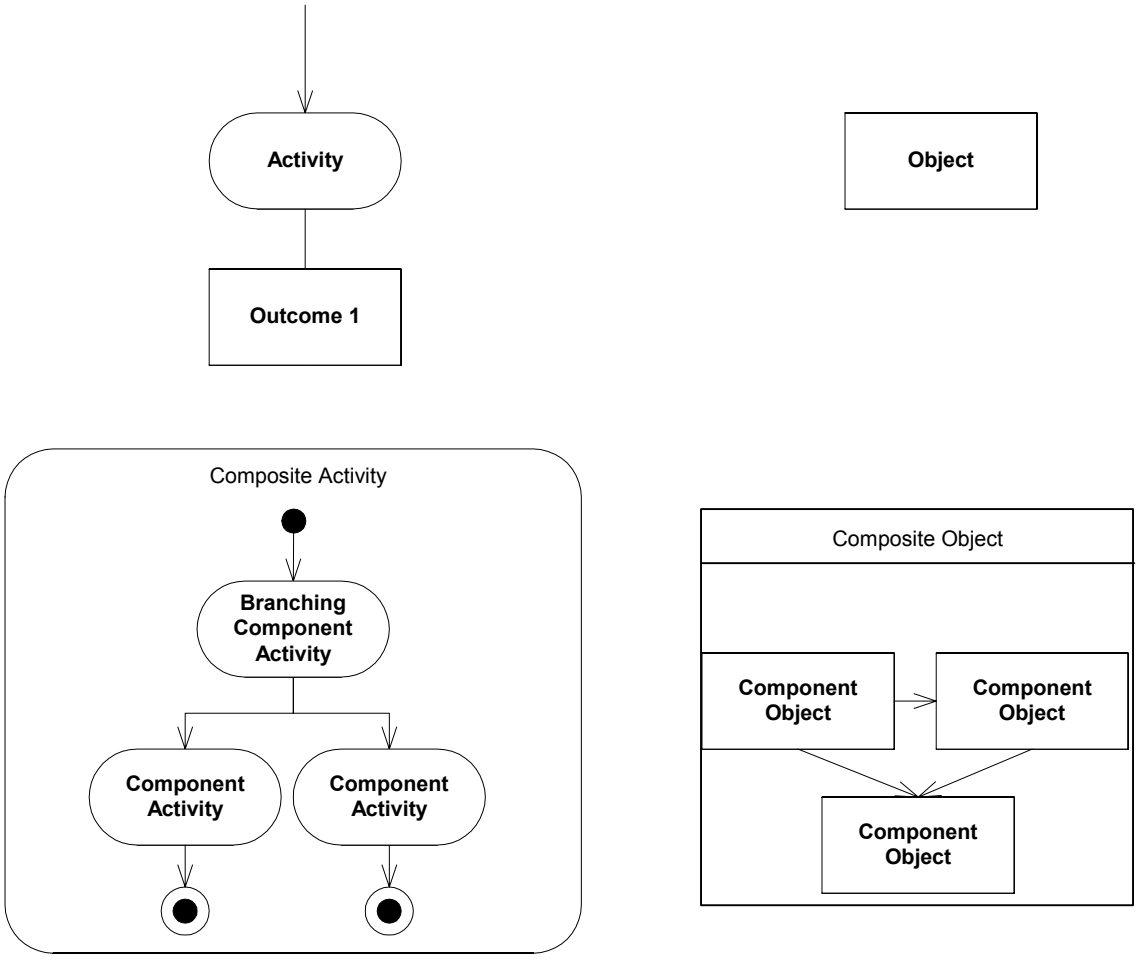

Fig. 3. RM-ODP activities

The view transformation between an atomic view and a composite view of a concept, be it for an activity or an object, is done by considering the constraints that govern the concept. The policies governing an activity are referred to whenever an activity is performed. In other words, the execution of the rule is triggered when the activity to which it is related is performed. These policies can be viewed in general as a prescription to do something, check, log, send, alert, stop, deny, authorize etc. which means that some entity needs to perform some sub-activity to guarantee that these policies are enforced. This gives us the component-activities comprising the graph defined by RM-ODP.

Terminology

In our approach we will be modeling an object or an activity in its atomic view when we want to consider the abstract view or in its composite view when we want to refine it. When we model the activity of an atomic object we will refer to it as a use case because it describes the behavior of the object as a system interacting with its environment. When we model the activity of a set of interacting component objects we will refer to it as collaboration. This is in accordance with the UML definition of a collaboration, that is "a set of participants and relationships that are meaningful for a given set of purposes." In our models we will distinguish between two types of policies. Actor Policies will be attached to individual actors' use cases while community policies will be attached to collaborations.

Goal and Policy Driven Design

There are 2 recurrent types of tasks that we perform when we design systems:

1. We take an object as an atomic object and we develop its view as a composite object. This includes two steps:
    a. We identify the goals of a community of actors and the community policies that govern the activities they engage in, in order to achieve these goals. The actors are the component-objects of the object of interest.
    b. We transform the community policies and goals into actor policies and goals.
    By doing this, we understand who does things in the system.
2. We take an activity as an atomic activity and we develop its view as a composite activity. This is the decision of what to do in the system. This could be done on an activity for a component object or on an activity for a composite object.

A complete model is the combination of models produced by each task. This complete model defines who does what in the system. In our models we use the <<trace>> stereotype to show the connections between models.

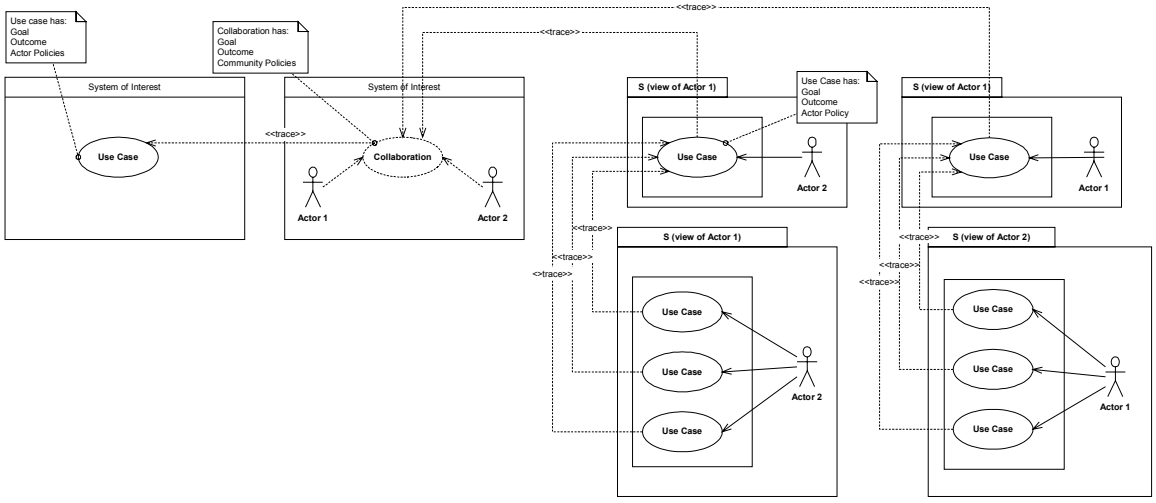

Fig. 4. Goal and Policy driven design

Figure 4 shows the overall approach we call Goal and Policy driven design. In order to understand it we will walk through the construction of this model. After having walked through the construction of each diagram of the model, it would interesting for the reader to come back to Figure 4 and see the relationships between the diagrams, expressed as <<trace>> relationships.

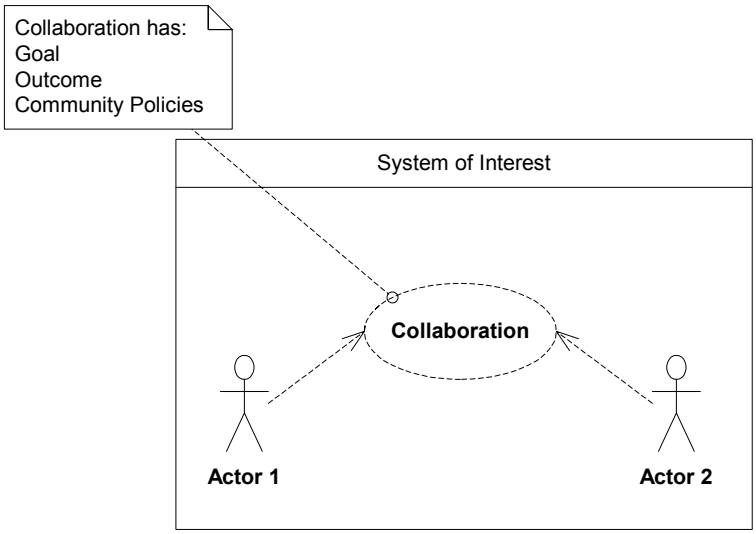

Fig. 5. The Organization as a collaboration

The modeling of an organization usually begins with the definition of the different actors within the organization and the collaborations they are involved in. Figure 5 shows the organization as a collaboration between actors having a goal and several policies governing the collaboration. Notice that the customer would typically be viewed as being inside the organization, which is a convenient way of not forgetting about their goals. The collaboration has a goal and is governed by a set of rules that we call community policies.

As we've seen in our discussion the Catalysis joint actions, considering the collaboration as a concept, which is external to its actors, prevents the modeler from considering the individual actors' goals. In order to view the actors' goals we need view the collaboration from the point of view of each one of the actors. We take a decision on who are the actors in the system. Thus, we move to the next view (Figure 6) where each actor is considered with its part in the collaboration. This transformation allows us to name the activities (use cases) within each actor and with the actor's vocabulary. When we distribute the collaboration into the different actors, we in fact define a set of use cases for each actor. These use cases represent the actor's part in the collaboration. The set of all actors' use cases together represent the collaboration. The community policies that "guide" the collaboration exist in some format within each actor's use case. We call these actor policies. We say that the actor policies trace the community policies.

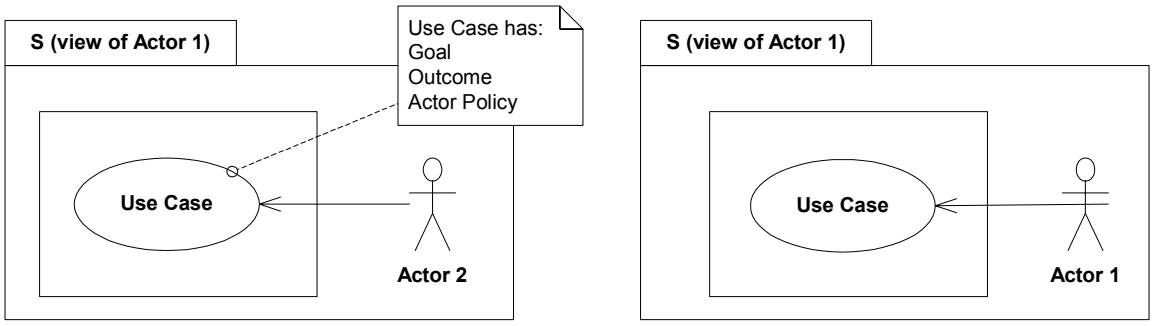

Fig. 6. The organization as a set of interacting actors

It is important to note that with this transformation we have not changed the nature of the collaboration but simply described it within the context of each actor. This actor centric view is equivalent to the collaboration centric view.

A use case is traditionally seen as a system centric description of the system with a set of actors. This translates well into our representation if we accept the view that every participant in a collaboration might become a system of interest. This is a generalization of the way we usually think of a system, i.e. the computer or IT system. In our approach the system can be any actor. The system's use case describes the system's behavior and its interactions with its environment. If we write a use case for the IT system, the environment is comprised of its users and other systems with which it interacts. If we write a use case for a user, the user is the system and the IT system and other people comprise its environments.

We can now refine the actor's use case, which is an activity into a composite activity. This refinement is done by considering the use case's policies, which impose constraints on the execution of the use case thereby defining the actor's sub-goals that will be carried out by the component-activities. There are many types of policies such as policies that impose invariants on the system, security related policies such as access control, obligation policies, etc. In most cases a policy can be seen as a need for some entity to take action when the activity to which the policy relates is executed. For example, an access control policy that says that some people are not allowed to access some information can be viewed as a goal of some entity in the system to check people's credentials and authorize or deny the access. A policy that guarantees an invariant such as ATM customers should not obtain more cash than is available in their account can be translated into a goal of some entity within the ATM to verify that the amount requested by a customer does not exceed the amount available in their account.

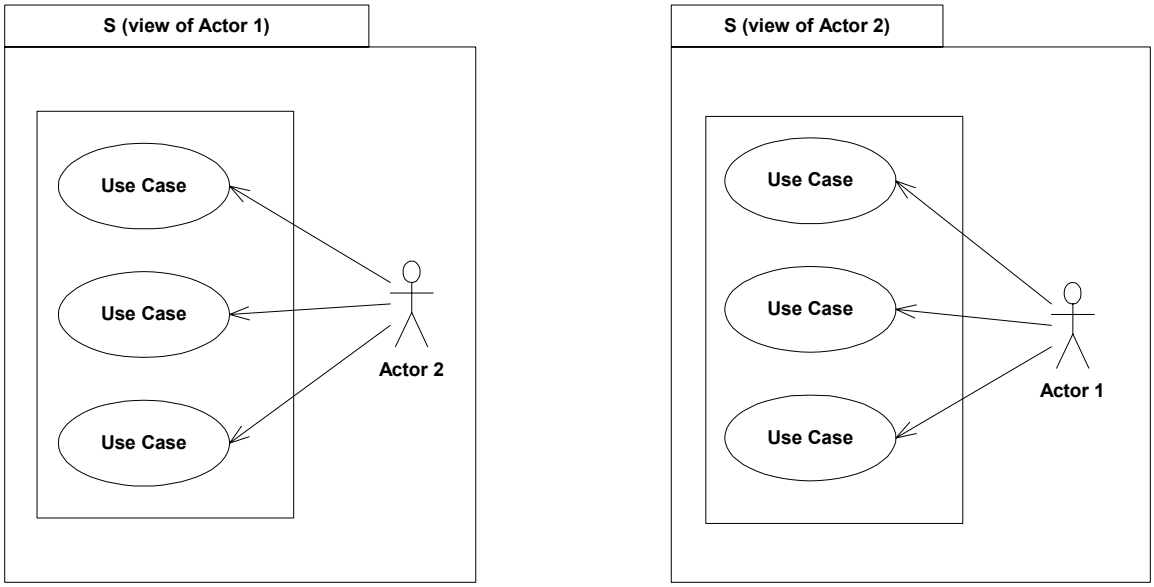

Fig. 7. An actor with its composite activity

The approach we have just described appears to be a purely top-down process. But it can also be used as a bottom-up process or better yet, in mixed mode. We know that people are likely to start modeling at some level of the organizational hierarchy and will probably expand downward and upward as keep on modeling the learning about their model [10]. Our approach supports this mixed process by enabling the modelers to ask "what are the goals associated with this policy and which actor needs to implement it?" and also "why does this actor have this goal?" which then defines a policy at the upper organizational level.

If we ask this kind of questions for the first diagram in Figure 5. We can, for instance, ask why is the organization engaged in this collaboration and why does the collaboration have this goal. We then generally find a model such as the one in Figure 8 where we see the organization as doing something. This activity of doing something can be expressed in a use case, which has a goal and actor policies. If this diagram represents for instance, an enterprise's topmost level, the goal would probably resemble a mission

statement. It is interesting to note that the outcome of the use case may not match the goal because the outcome of an enterprise is eventually its demise while this is not its goal.

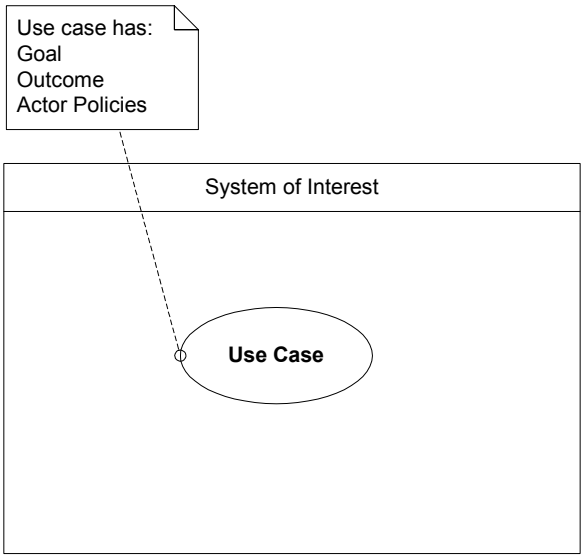

Fig. 8. The Organization as an activity

Example

We can now apply our approach to the example of a store (Figure 9). At the highest organizational level we simply say that the store exists. It has the goal of continuing to exist. It is governed by an actor policy that says that is has to sell merchandize in exchange for money. We can now identify 2 actors within the store, a customer and a vendor who are related by the Sale collaboration. This collaboration is defined by the post condition: Goods delivered to Customer in Exchange for Money. The community policies guarantee the fairness of the exchange:

- Customer must pay for goods
- Goods must be delivered to customer

Defining the sale in the context of each one of the actors, define the following use cases:

- Customer.Buy
- Vendor.Sell

Customer.Buy has the following actor policies:

- Customer must pay for goods.
- Customer must accept delivered goods

Vendor.Sell has the following actor policies:

- Vendor must deliver goods.
- Vendor must get paid

So we now have 2 sub-goals for the Customer, which are:

- Pay for the goods
- Receive the delivered goods

And 2 sub-goals for the vendor:

- Get Paid
- Deliver Goods

This example shows that by defining the goal of the collaboration between a vendor and a customer in terms of the goals of the vendor and customer, we can define the policies that constrain each of these actors. These policies gave us 4 sub-goals, 2 for each actor. Evidently, we can go on and find more sub-goals for, say, pay, or take delivery. However, as noted by Weinberg [20], "*heuristics don't tell you when to stop*." Our approach is not an exception. Using it down to the smallest sub goals will produce a behavioral model, which is probably over specified and unusable. It would be useless, to specify sub goals that are too detailed such as for example, object needs to be created before it can be used, unless it is important for the model to be understandable. It is up to the modeler to know when to stop depending on their goals and constraints.

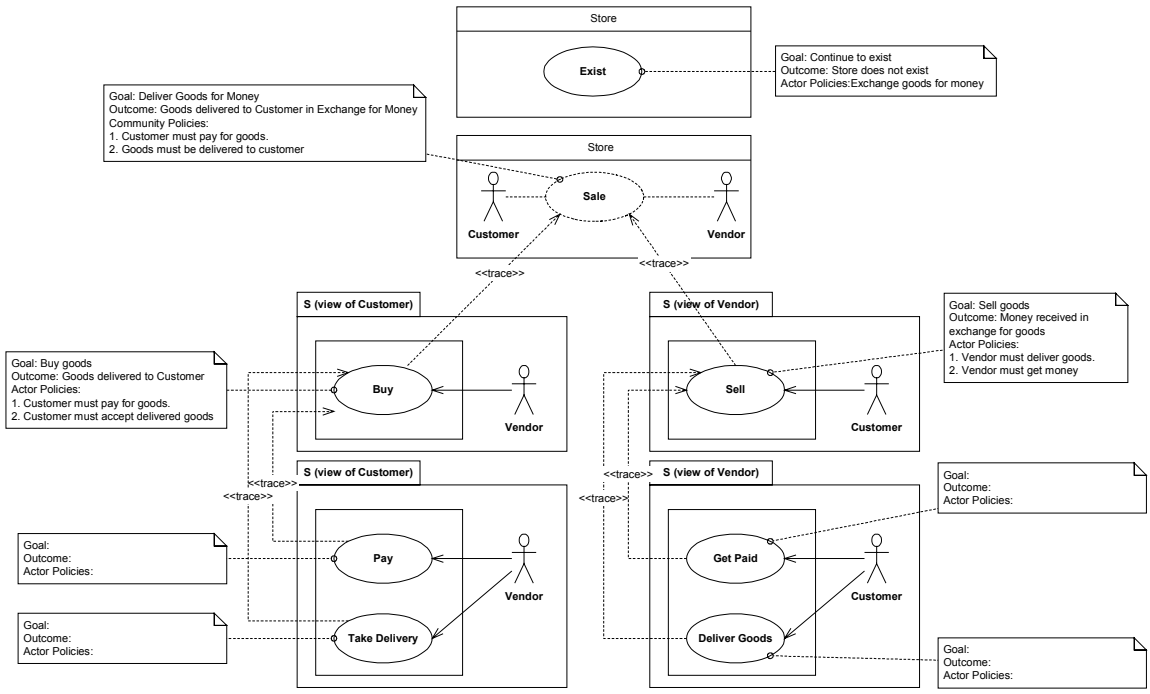

Fig. 9. The store example

Conclusions

In this paper we have described a design approach based on the dual consideration of actors' goals and business rules. We have showed how defining actors' goals within their context enables us to formulate the policies that govern the activities related to the achievement of these goals. These rules in turn help us formulate new goals to be implemented by actors and so on.

We have thus merged the seemingly separate domains of goals and business rules. The result is an integrated process that enables development teams to not only produce models that take into account actors' goals and constraints but also help them in defining these goals and tracing goals to the reasons actors may have for formulating these goals. Knowing the relationship that exists between goals and rules may help us to challenge the way we do things and the goals that we pursue when the rules or goals change.

We have defined and refined this approach on several case studies such as a point of sale and a video store, which demonstrates its feasibility and value in making design decisions explicit. We also noticed that a CASE tool support would significantly help with the application of the approach to more complex systems.

References

1. Bock, C., Goal-Driven Modeling, Journal of Object-Oriented Programming. September (2000), Vol. 13, No. 5.
2. Cockburn, A., Writing Effective Use Cases. Addison-Wesley, Reading, MA (2000)
3. Constantine L., L., Lockwood, L., Software for Use. ACM Press NY, NY (1999)
4. D'Souza D. F., Wills A. C., Objects, Components, and Frameworks with UML – The Catalysis Approach. Addison-Wesley, Reading, MA (1999)
5. Gottesdiener E., Capturing Business Rules. In: Ambler, S. W., and Constantine L. L., The Unified Process Inception Phase: Best practices in Implementing the UP. CMP Books, Lawrence, Kansas (2000)
6. ISO/IEC ITU-T: Open Distributed Processing – Basic Reference Model – Part 2: Foundations. Standard 10746-2, Recommendation X.902 (1995)
7. Jacobson, I., Booch, G., Rumbaugh, J., The Unified Software Development Process. Addison-Wesley, Reading, MA (1999)
8. Kilov, H., Business Specifications: The Key to Successful Software Engineering. Upper Saddle River, NJ (1999)
9. Kuutti, K. Activity Theory and its applications in information systems research and design. In: Nissen, H.-E., Klein, H.K. and Hirschheim, R. (eds.) Information Systems Research Arena of the 90 's. North-Holland, Amsterdam (1991), pp. 529-550.
10. Lakoff, G., Women, Fire and Dangerous Things – What Categories Reveal about the Mind. Chicago Press (1987)
11. Larman, C., Applying UML and Patterns. Prentice Hall, Upper Saddle River, NJ (1998)
12. Mylopoulos, J., Chung, L., Yu, E., From Object-Oriented to Goal-Oriented. Communications of the ACM, January (1999), vol. 42. No. 1.
13. Le Moigne, J-L, La modélisation des systèmes complexes. Dunod, Paris (1990)
14. Robertson S., Robertson J., Mastering the Requirements Process. Addison-Wesley, Reading MA (1999)
15. Sloman, M., Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Plenum Press. (1994), Vol. 2 No 4
16. OMG: Unified Modeling Language Specification. Version 1.3 (1999) (www.omg.org)
17. Von Bertalanffy, L, General System Theory. George Braziller, NY, NY (1969)
18. Wegmann, A., Genilloud, G., The Role of "Roles" in Use Case Diagrams**.** Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York, pp. 210 – 224 (2000)
19. Wegmann, A., Naumenko, A., Conceptual Representation of Complex Systems Based on RM-ODP as an Ontology. *EPFL-DSC Technical Report* (2001)
20. Weinberg, G. M., An Introduction to General Systems Thinking. Wiley & Sons, NY, NY (1975).